1. Document Structure & Chunking Logic

The provided document (~10,500+ words) was preprocessed and segmented before use in the chatbot.

- **Cleaning:** Removed headers, footers, repeated "Page X of Y", multiple newlines, and redundant whitespace.
- **Sentence tokenization:** Used **NLTK Punkt tokenizer** to split text into sentences.
- **Chunking strategy:**
    - Targeted **100–300 words per chunk**, ensuring sentence boundaries are respected.
    - If a sentence exceeded the max length (rare), it formed its own chunk.
    - Added optional **overlap of ~30 words** between chunks to reduce boundary information loss.
- **Output:** Final dataset contained ~N chunks (exact number depends on document). Each chunk stored with id, text, word count, and sentence indices.

This ensures that retrieval covers the document while keeping context sizes manageable for the LLM.

2. Embedding Model & Vector Database

- **Embedding model:** all-MiniLM-L6-v2
    - Compact (~22M params), fast inference, good tradeoff between accuracy and efficiency.
    - Generates **384-dimensional embeddings**.
- **Vector database:**
    - **FAISS IndexFlatIP** with L2-normalized vectors → approximates **cosine similarity**.
    - ChromaDB was tested as an alternative but FAISS was chosen for efficiency and simplicity.
- **Persistence:**
    - Embeddings stored in vectordb/embeddings.npy.
    - Metadata (including full chunk text) stored in vectordb/metadata.json.
    - FAISS index saved in vectordb/index.faiss.

- # 3. Prompt Format & Generation Logic

- The RAG pipeline injects retrieved chunks into a fixed prompt template before querying the LLM.
- **Prompt Template:**

  You are a helpful assistant. Answer the USER QUESTION strictly using the SOURCES below.

  If the answer is not in the sources, say "I don't know."

  SOURCES:

  [1] <chunk text>

  [2] <chunk text>

  ...

  USER QUESTION:

  {query}

  Answer with citations like [1], [2] where appropriate.

- Retrieved top-k chunks (k=3 by default) are concatenated with source IDs.
- User query is appended at the end.
- The model is instructed to only answer from sources and provide inline citations.

Model: mistralai/Mistral-7B-Instruct-v0.2 (can be swapped for Zephyr/LLaMA/Flan-T5 depending on compute).
Streaming: Implemented with Hugging Face TextIteratorStreamer in a background thread, so tokens are displayed incrementally in Streamlit.

. Example Queries & ResultsSuccess Cases

1. **Q:** What is the purpose of the chatbot?
   A: The chatbot is designed to answer user queries based on a provided document set, using a RAG pipeline [1].
2. **Q:** How are chunks generated from the document?
   A: The text is split into segments of 100–300 words using sentence-aware splitting, with optional overlap [2].
3. **Q:** Which vector database options are supported?
   A: The system supports FAISS, Chroma, or Qdrant for semantic retrieval [3].

Failure / Edge Cases

4. **Q:** Who founded Amlgo Labs?
   A: *"I don't know."* (Correct, since this info is not in the provided document).
5. **Q:** Can this chatbot summarize a new PDF uploaded at runtime?
   A: The model attempted to guess, sometimes stating "yes, it can" — this is a **hallucination**, since the current implementation requires preprocessing before new documents can be queried.

5. Notes on Limitations

- **Hallucinations:**
  - Although the prompt reduces hallucinations, the model may still fabricate minor details.
  - Mitigated by strict instructions and low temperature (0.0).
- **Performance:**
  - **Chunking/embedding step**: ~2–3 minutes on CPU for 10k+ words.
  - **Retrieval**: <100ms with FAISS (on CPU).
  - **Generation latency**:
    - ~3–5 seconds for short answers on GPU.
    - Much slower on CPU for 7B models; recommended to use smaller models if no GPU is available.
- **Context size limits:**
  - Large documents require careful chunking to avoid exceeding LLM input limits.
- **Scalability:**
  - Works well for single documents. For multiple/large corpora, a hierarchical retriever or re-ranking would improve results.

. Conclusion

This project demonstrates an end-to-end **Retrieval-Augmented Generation (RAG) chatbot**:

- Document preprocessing → chunking → embeddings → vector DB → retriever → LLM with streaming.

- Achieves **grounded answers with citations** and **real-time interactive interface** via Streamlit.
- Flexible: can switch between FAISS/Chroma, different embedding models, or different LLMs depending on resources.

Future improvements:

- Add **cross-encoder re-ranking** for better retrieval.
- Support **dynamic document uploads** with incremental indexing.
- Explore **quantized models** (e.g., 4-bit Mistral) for faster local inference.